Automatized Measurement Setup for Performance Evaluation of Wi-Fi Mesh Networks (A.M.S.P.E.W.M.N.)

A Thesis submitted by David Torres Alcantud for the degree of Electronic Systems Engineering in Telecommunications of the Universitat Politècnica de Catalunya (UPC)

Supervised by:
Santiago Silvestre Berges
Philipp Svoboda

Vienna, October 2017
Abstract

Telecommunications is an everchanging world. Everyday we are faced with improvements in the way we communicate and so in the tools that we use, especially in the field of Wi-Fi connection. The 5G connection will soon start functioning even though its former frequency, the 4G connection, started to work not many years ago.

Currently, the field of robotics plays an important part in measuring the parameters of Wi-Fi signals in places where humans cannot access. The 5G experiment happens in an anechoic chamber, which during the experiment has to be closed and no radiating element can be inside as it would alter the results of the test. it is for that, that the new technologies require robots as the instrumentation for measuring.

At first, we decided on the type of robot that would attain the best results based on its characteristics and which tests we would carry out during the experiment.

After, we had to agree on which components (sensors, microprocessor, etc.) would be the best to perform each task.

At this point, it was the moment to set the schedule for the project. Which consisted on, first, assembling the components that would shape the hardware and finishing with the programming of the software, which had to decide the best alternative in any given scenario. During the project we were faced with issues that we fixed in a different way than expected.

At the end, we tested the performance of the robot in different scenarios. With the results obtained in the tests we could figure out ways of improving its performance.
Dedication

I would like to dedicate this project to everyone who helped me directly or indirectly. Those people made possible this project.

Especially, I want to dedicate this work to my parents Arturo and Rosario who never let me give up the degree, for their support, their advice and for taking care of me the whole time.

My brother Rubén for his unconditional love and support.

My girlfriend Lorena who was always by my side and encouraged me to carry on.

My friend Sandra who helped me whenever I needed without complaining.

My university colleagues with whom I spent many hours studying and gave me some of the best moments during my time at the university.

And last, but not least, my supervisors, Philipp Svoboda and Santiago Silvestre Berges, thanks to whom I had the opportunity to test my knowledge and finish the degree and finally become an engineer.

In general, this work has been possible thanks to the people who have been by my side during these five years.
Acknowledgements

I am grateful to all the people that were there these five years and particularly to those who were there during the Erasmus.

First, thanks to Dr. Philipp Svoboda and Dr. Santiago Silvestre Berges for their extraordinary support and availability in this project acting both as my supervisors and sharing their knowledge with me.

This project would have been impossible without the support of the Technology University of Vienna which allowed me to develop my work in their laboratories and provided all the material that I required. Also, to Universitat Politècnica de Catalunya where I studied and learnt about Telecommunications.

Thanks to my university colleagues, my family and my girlfriend for their support, friendship and the good moments.
Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>08/05/2017</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>10/07/2017</td>
<td>Document revision</td>
</tr>
<tr>
<td>2</td>
<td>06/10/2017</td>
<td>Document revision</td>
</tr>
</tbody>
</table>

Document distribution list

<table>
<thead>
<tr>
<th>Name</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>David Torres Alcantud</td>
<td><a href="mailto:david.torres.alcantud@gmail.com">david.torres.alcantud@gmail.com</a></td>
</tr>
<tr>
<td>Santiago Silvestre Berges (Spain)</td>
<td><a href="mailto:santiago.silvestre@upc.edu">santiago.silvestre@upc.edu</a></td>
</tr>
<tr>
<td>Philipp Svoboda (Vienna)</td>
<td><a href="mailto:psvoboda@nt.tuwien.ac.at">psvoboda@nt.tuwien.ac.at</a></td>
</tr>
</tbody>
</table>

Written by: David Torres Alcantud
Reviewed and approved by: Santiago Silvestre Berges

<table>
<thead>
<tr>
<th>Written by: David Torres Alcantud</th>
<th>Reviewed and approved by: Santiago Silvestre Berges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date 06/10/2017</td>
<td>Date 06/10/2017</td>
</tr>
<tr>
<td>Name David Torres</td>
<td>Name Santiago Silvestre</td>
</tr>
<tr>
<td>Position Project author</td>
<td>Position Project supervisor</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction .................................................. 1
   1.1 Goals .................................................................. 1
   1.2 Requirements .................................................. 2
   1.3 Document structure ............................................ 2

2 Line follower robot .............................................. 4
   2.1 Architecture and basis ......................................... 4
      2.1.1 Components .............................................. 5
      2.1.2 Line follower behavior ................................... 7
      2.1.3 Functionality .............................................. 8
   2.2 Line detector ................................................... 9
      2.2.1 Camera and signal processing ......................... 9
   2.3 Assembly and integration .................................... 15
      2.3.1 Chassis and body ....................................... 15
      2.3.2 Motor driver ............................................ 15
      2.3.3 Microcontroller ....................................... 16
      2.3.4 Camera .................................................. 16
      2.3.5 Actuators (Motors and wheels) ....................... 17
      2.3.6 Batteries ................................................. 18
      2.3.7 Power supply Raspberry Pi ........................... 18
      2.3.8 Connections ............................................ 19
   2.4 Line follower behavior ....................................... 20
   2.5 PID Tracking Control Algorithm ......................... 22

3 Data acquisition ............................................... 23
   3.1 Signal detection .............................................. 24

4 Communication user - robot .................................. 25
   4.1 TCP/IP model ................................................. 25
   4.2 Graphic user interface ...................................... 26

5 Budget .................................................................. 28

6 Conclusions and future work .................................. 31
   6.1 Conclusions ................................................... 31
      6.1.1 Line Follower Robot .................................... 31
      6.1.2 Data acquisition ....................................... 31
      6.1.3 Graphic User Interface ............................... 31
6.2 Future work ................................................. 32
  6.2.1 Line Follower Robot .................................. 32
  6.2.2 Data acquisition .................................. 32
  6.2.3 Graphic User Interface ........................... 33

A LAB Color Space ........................................ iv
B Ramer Douglas Peucker algorithm ........................ v
C Proportional Integral Derivative control .................. vii
D UDP Protocol vs TCP Protocol ........................... ix
E Line Follower Robot ...................................... xi
F Detection of an object .................................. xvi
G Detection of shapes on the track ......................... xviii
H Detection of color on the track ........................ xx
I Data acquisition ........................................ xxii
J Communication User - Laptop .......................... xxvii
K Line follower Robot multi threading .................. xxx
L Data acquisition Robot multi threading ............... xxxi
M Graphic user interface ................................. xxxii
## List of Figures

1.1 Anechoic chamber .................................................. 1
2.1 Line follower robot .................................................. 4
2.2 How IR sensor works .................................................. 5
2.3 How line follower robot works ..................................... 7
2.4 Wi-Fi Coverage study ............................................... 8
2.5 Frame obtained by the camera .................................... 9
2.6 Region of interest .................................................... 9
2.7 Grey scale ROI ......................................................... 10
2.8 Guassian filter to blur the frame .................................. 10
2.9 Back ground substractor ............................................ 10
2.10 Invert the pixels of the image .................................... 11
2.11 Threshold to remove the noise .................................... 11
2.12 Invert the pixels of the image .................................... 11
2.13 Erode the image ..................................................... 12
2.14 Dilate the image ..................................................... 12
2.15 Frame with the contours detected .............................. 13
2.16 Frame with the contours detected .............................. 13
2.17 Detection of the shape and colour ............................... 14
2.18 Car structure ......................................................... 15
2.19 L298N H-bridge ..................................................... 15
2.20 Raspberry Pi model 3B .............................................. 16
2.21 Camera USB connected to Raspberry ......................... 16
2.22 DC motor and wheel ............................................... 17
2.23 Tower pro microservo MG995 .................................... 17
2.24 18650 3.7V Li-ion Battery ...................................... 18
2.25 Step-down DC-DC converter module ......................... 18
2.26 Car design high angle ............................................ 19
2.27 Car design rear angle ............................................. 19
2.28 Car design frontal angle ......................................... 19
2.29 Line follower robot flowchart ................................. 20
2.30 How calculate the angle .......................................... 21
2.31 PID control LFR ..................................................... 22
3.1 iwlist command ...................................................... 24
4.1 Frame with the contours detected .............................. 25
4.2 Graphic user interface .............................................. 26
5.1 Gantt diagram .................................................. 30
A.1 CIELAB color space ........................................ iv
B.1 Ramer–Douglas–Peucker algorithm ........................ v
C.1 Scheme of a PID Control .................................. vii
List of Tables

2.1 Theoretical states of the LFR. ........................................... 7
5.1 Cost of the components. .................................................. 29
D.1 UDP vs TCP. ................................................................. x
Chapter 1

Introduction

Telecommunications is a field where a short period of time means a lot of changes, it is for that, that in few years we find huge improvements. For example, data transmission, which one of its areas of study is Wi-Fi and its qualities, is now close to working with 5G.

The theories about data transmission need to be tested, so it is necessary to create some gadget that can perform the tasks of measuring signals. There are many ways for us to check those theories. One of them is the use of robots that can capture the environmental signals and process those signals to come up with results.

1.1 Goals

The intention of this project was to design a method to analyze Wi-Fi signals in specific scenarios. The place where the experiment would take place was an anechoic chamber\(^1\). In this room no human activity is allowed once the process starts. This project arised from the necessity to find a way to perform the measurements with no radiating objects in the room.

Taking into account the previous facts, we had to design and assembly a line-follower robot able to make Wi-Fi measurements while tracking a black line painted on the floor. During the experiment, the robot had to save the information in a data base to generate a Wi-Fi coverage study.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{anechoic_chamber.png}
\caption{Anechoic chamber}
\end{figure}

\(^1\)Is a room designed to completely absorb reflections of either sound or electromagnetic waves.
1.2 Requirements

A reliable Wi-Fi coverage study lasts for at least six hours, that means that the battery had to last longer than six hours.

The measuring device needed an component to acquire the environmental signals during the circuit.

Moreover, the user could use a GUI to modify from the speed to turn it on or off. The GUI had to show the Wi-Fi information in real time (name of the signal, power transmission, etc.). Afterwards, that information was sent to a database to process it and generate a coverage of the study.

1.3 Document structure

This document is divided in five blocks. The first block describes the line follower mechanism (how to assemble a simple model, the components that compose it, how to follow the circuit, etc.). The second explains the Wi-Fi data acquisition (tools used to get the information, the script which filters the information, etc.). The third shows the communication between the robot and the user. The fourth shows how to organize the project and the cost. And the last one presents the results and ways to improve the robot’s performance.

The first block is divided in six sub blocks. The first, it explains how we designed a simple line-follower robot, step by step. Also, we compare the different technologies that we can use to implement all the parts of the device. The second, it shows how using a USB camera the robot can follow the track. The third sub block describes the hardware used in the project, the characteristics and the reasons why we chose each component. Furthermore, it illustrates the steps to assembly every item and create the final design. The fourth, it illustrates the performance of the robot, the algorithm that decides the future movement. The last sub block explains the control of the robot, which uses a loop to correct the robot’s miscalculation.

The second block describes the process to obtain the information of the Wi-Fi signal. To obtain this information, first, we need to use a peripheral to detect the signals. Once the signal is detected, the information is processed to acquire the useful parameters and save them in the data base.

The third block is divided in two sub blocks. The first sub block depicts the protocols that we use to establish a communication between the user and the robot. The second shows the GUI, also explaining all the options from which the user benefits when using the robot.
CHAPTER 1. INTRODUCTION

The fourth block describes the organization of the project. It displays all the components needed to assemble the robot. Moreover, it includes the price of building one robot. We also compare the price of producing one unit to building one hundred units. Additionally, using the schedule of each task and looking for the average salary in Spain, we can estimate how much it would cost to pay the workers. Finally, it breaks down the cost of the whole project.

The last block summarizes all the work done, explains the tests done in each section and the conclusions. With the results obtained we can figure out the way to enhance the robot’s performance.
Chapter 2

Line follower robot

A line-follower robot [1] is basically a robot designed to follow a line or path already predetermined by the user. This line or path may be as simple as a physical white line drawn on the floor or as complex as path marking schemes. This type of robot is easy to make, the complexity of the device is on the imagination of the designer. There are a wide variety of applications for this type of machine.

2.1 Architecture and basis

Generally, the LFR ¹ is one of the self-operating mobile machines that follow a line drawn on the floor. The path can be a visible black line on a white surface or vice versa. The basic operations of the line-follower are as follows:

- Capturing the position of the robot, relative to the line, with sensors mounted at the front of the robot. It could use a camera instead of optical sensors to obtain the highest resolution and robustness possible.

- Steering the robot to track the line with any steering mechanism, the robot has the option of being steered by the user.

- Controlling the speed according to the lane condition, there is the possibility for the user to regulate the speed within limits. The speed is limited during passing a curve due to the friction of the tire and the floor.

![Figure 2.1: Line follower robot.](image)

¹Line-Follower Robot.
This kind of robot can be used for many different purpose. Because of that, there is no specific design. That means that, the robot can have a specific functions depending on the design.

2.1.1 Components

As there are a lot of possible designs, this section is about the required parts that need to be assembled.

- Sensors
  They are the most important component because the robot needs to recognize the line, in great measure this depends on the sensors. The most common line detector is the IR sensor\(^2\). The sensor consists of two diode. One diode is the transceiver and the other the receiver.

![Figure 2.2: How IR sensor works.](image)

The received signal depends on the distance between the IR led and the surface. Also, it is important the colour of the surface, a white surface reflects better the signal than a black one. This is the reasons why the line has to be black on a white floor or vice versa.

On the other hand, there is the possibility to use a camera to follow the track. This option is not commonly used due that setting up a camera is harder than setting up sensors, although depending on the tasks it has to perform could be a good option

\(^2\)InfraRed sensor.
• Power supply
The robot is a mobile vehicle, that means that the power supply has to be mobile. There are different mobile alternatives. For example, single batteries, plural batteries or solar panels. The design and the power of the battery depends of the performance of the robot. The important factors to take into account are:

- Size
- Capacity
- Voltage

- Chemistry
- Life cycle
- Watts and Volt-amps

- Specific energy and energy destiny
- Specific power
- Load

• Chassis
The robot is mainly constituted by its chassis. They can be found in different shapes, materials, etc. The goal here it is to ensure the robot’s performance and carry all the components. It is important to choose the right structure because it influences the robot’s results.

• Actuators
It is the part responsible for the robot’s movement. The commonly used in this kind of robots are:

- DC motor: It is a continuous rotation motors. Once the motor starts it is working until the power is off. The speed of DC motor is regulated using pulse width modulation.
- Servo motor: It is formed with four things: a DC motor, a gearing set, a control circuit and a position-sensor. The position of the motor is more precisely than a DC motor. The angle of rotation is from 0 to 180 degrees. PWM\textsuperscript{3} is used for the control signal, the duration of the pulse determines the position.
- Stepper motor: It is a servo motor that uses a different method of motorization. This motor uses multiple toothed electromagnets arranged around a central gear to define position. Stepper motors need an external control circuit or microcontroller.

• Control
As said before, every design has a purpose. Depending on the components you choose and their usefulness, we use one type of control or another. From a analog/digital adapter to a microprocessor.

\textsuperscript{3}Pulse Width Modulation.
2.1.2 Line follower behavior

Following the previous sections, we are ready to make our line follower robot. This section explains the easiest design possible. The robot has two IR sensors and depending on the inputs the robot will have a determined reaction.

There are two sensors, that means that the robot has four possible states.

<table>
<thead>
<tr>
<th>Left sensor</th>
<th>Right sensor</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>It doesn’t detect line. It is stopped until the sensors detect a track.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>It detects only the right, that means the robot is going to the left. It turns to the right.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>It detects only the left, that means the robot is going to the right. It turns to the left.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>It detects both sensors. It keeps straight until the sensors detect another possible state.</td>
</tr>
</tbody>
</table>

Table 2.1: Theoretical states of the LFR.

The table is represented in the next picture.

![Diagram showing how the line follower robot works](image)

Figure 2.3: How line follower robot works.

The easiest model is limited because it only has two sensors and that makes the robot unable to detect sharp curves. Also, as it is a simple version there is no adjustment with the wheels, when only one sensor detects the track the robot stops one wheel while the other keeps moving that provoking it to turn.
2.1.3 Functionality

In the field of telecommunications there is the necessity to make measurements when theoretical hypotheses are tested. These tasks are performed by robots. They are able to realise measurements during the test.

One of the possible options is the line follower robot. It is able to interact with the user who can choose the route and establish strategic points to make more measurements than in other points. These robots are equipped with antennae and sensors to make all the measurements.

Figure 2.4: Wi-Fi Coverage study.
2.2 Line detector

One of the possible options is the line follower robot. It is able to interact with the user who can choose the route and establish strategic points to make more measurements than in other points. These robots are equipped with antennae and sensors to make all the measurements.

2.2.1 Camera and signal processing

LFR uses image processing through a USB camera to find the path. The camera captures an image of its surroundings, then it searches for the black tape which is stuck to the ground. The image taken by the camera is modified by the Python library called OpenCV\[2, 3\], changing the size and the rate of the frames. Once the picture has been modified, the program processes every frame following the next steps.

![Frame obtained by the camera.](image)

Once the program has got the image, to accelerate its processing and make it more precise, the code uses the ROI$^4$.

1. Using $roi = frame[10:2*frame.shape[0]/3, frame.shape[0]/12:frame.shape[1]-20]$ we obtained the ROI.

![Region of interest.](image)

---

$^4$Region Of Interest.
2. Using `cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)`, the function converted an input image from one colour space to another. In case of a transformation to/from RGB colour space, the order of the channels should be specified explicitly (RGB or BGR). To change from RGB to $Y$ (gray scale) the function uses the next formula.

$$ Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (2.1) $$

![Grey scale ROI](image)

Figure 2.7: Grey scale ROI.

3. Using `cv2.GaussianBlur(grayscale, (5,5), 0)`, the function blurred the image convolving the image with a Gaussian low-pass filter. Gaussian blurring is highly effective in removing Gaussian noise from the image.

![Guassian filter to blur the frame](image)

Figure 2.8: Guassian filter to blur the frame.

4. Using `cv2.BackgroundSubtractorMOG().apply(blur, leraningRate = 0.001)`, the function modeled each background pixel by a mixture of $K$ Gaussian distributions ($K = 3$ to $5$). The probable background colours are the ones which stay longer and more static.

![Back ground substractor](image)

Figure 2.9: Back ground substractor.
5. Using threshold `cv2.threshold(fgmaxs, 0, 255, cv2.THRESH_BINARY_INV)`, the function changed the value of the pixels, if pixel value was greater than a threshold value, it was assigned one value (white), else it was assigned another value (black).

\[
dst(x, y) = \begin{cases} 
  \text{white} & \text{if } \text{input}(x, y) \geq \text{Threshold}(x, y) \\
  \text{black} & \text{otherwise}
\end{cases}
\] (2.2)

![Figure 2.10: Invert the pixels of the image.](image)

6. Using threshold `cv2.threshold(thresholdInvert1, 20, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)`, the function removed the noise of the image.

![Figure 2.11: Threshold to remove the noise](image)

7. Using `cv2.threshold(thresholdNoise, 0, 255, cv2.THRESH_BINARY_INV)` the function inverted the pixels of the image.

![Figure 2.12: Invert the pixels of the image.](image)
8. Using `cv2.erode(thresholdInvert2, np.ones((5,5), np.uint8), iterations = 1)`, the function eroded the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum was taken.

\[
dst(x, y) = \min_{(x', y')} \text{input}(x + x', y + y')
\]  

(2.3)

Figure 2.13: Erode the image

9. Using `cv2.dilate(erodeImg, np.ones((5,5), np.uint8), iterations = 1)`, the function eroded the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum was taken.

\[
dst(x, y) = \max_{(x', y')} \text{input}(x + x', y + y')
\]  

(2.4)

Figure 2.14: Dilate the image

10. Using `cv2.findContours(dilateImg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`, the function found the pixels with white colour from black background and joined all the continuous points doing a curve.

Besides the contours found it, we could calculated the area of every contour, with the function `cv2.moments(contour)`, and got the moments of the contour to calculated the centroids[4]:

\[
\begin{align*}
\text{centroid}_x &= \frac{M_{10}}{M_{00}} \\
\text{centroid}_y &= \frac{M_{01}}{M_{00}}
\end{align*}
\]  

(2.5)
CHAPTER 2. LINE FOLLOWER ROBOT

The centroids indicated where it was the position of the robot relative to the line.

![Figure 2.15: Frame with the contours detected](image)

Once the edges of the ROI are calculated, they overlap on the original image. This is the one used to decide which direction the robot takes.

![Figure 2.16: Frame with the contours detected](image)

Moreover, the robot has the capacity to detect special signs. The robot could detect rectangles in different colours[5]. If the user activated this mode, s/he could choose between the colours red, blue or green or no colour, any other option would be impossible to detect. When the robot detects a rectangle, it would stop for some time at that point.

This task was performed in two subtasks. First task uses a function that detects the edges of the shapes using the Ramer–Douglas–Peucker algorithm[6]. The second works with $L^*a^*b^*$ colour space[7] to calculate the colour of the shape.
When the user activates this mode, the robot stops during some time if it detects a rectangle on the chosen colour. Also, there is the option that the robot ignores the coloured shapes.

Figure 2.17: Detection of the shape and colour.
CHAPTER 2. LINE FOLLOWER ROBOT

2.3 Assembly and integration

Once the line detector was done, the next step was to make a structure able to follow the track. The next points explain the material used for this part of the project.

2.3.1 Chassis and body

Currently, there are different types of chassis for this kind of robot, the most important fact is the materials, such as plastic, aluminum and carbon-composites.

The project was to create a prototype of a mobile machine, it is for that, that we chose to use a small self-operating car able to carry all the necessary components. To guarantee that, the best option was to choose a plastic chassis due to the important restriction of the weight, as there were many circuits.

![Figure 2.18: Car structure](image)

2.3.2 Motor driver

The motor driver was a L298N H-bridge IC which was used to control two DC motors. It was a high voltage, high current, dual full bridge. It was the most common model, for that it was easy to find. The reason to choose this model was all the circuit in a small PCB\(^5\) which meant more space on the chassis.

![Figure 2.19: L298N H-bridge](image)

\(^5\)Printed Circuit Board
2.3.3 Microcontroller

We used a Raspberry Pi 3[8, 9], the microcontroller was the Atmel ATmega644, had a RISC architecture\(^6\), internal oscillators[10, 11], timers, USB, ADC, PWM channels[12], etc... With this microcontroller we could program in EEPROM\(^7\) and SRAM.

![Raspberry Pi model 3B](image)

Figure 2.20: Raspberry Pi model 3B

The idea of the final robot was that the robot would perform more tasks. Thinking in the future work, the robot had the last model of Raspberry Pi which was the one with more functions.

2.3.4 Camera

The first idea was to use a Raspberry camera, but after some problems with the connection between the board and the camera, the best solution was to use a USB camera which made easier the process of obtaining the images of the surroundings and send those images to the board.

![Camera USB connected to Raspberry](image)

Figure 2.21: Camera USB connected to Raspberry

---

\(^6\)Reduced Instruction Set Computer architecture(kind of CPU design)
\(^7\)Electrically Erasable Programmable Read-Only Memory
2.3.5 Actuators (Motors and wheels)

As mentioned above, the best option for this project was to use DC motors and servo motors.

- The DC motor was used to control the speed of the rear wheels, it was the simplest motor to perform the task and we decided to choose it.

![DC motor and wheel](image1)

Figure 2.22: DC motor and wheel

- The Servo motor can turn 180°, but this model can adjust the rotation angle to any position between 0 and 180°. This motor provides a better movement on the path because it rotates the necessary angle.

![Tower pro microservo MG995](image2)

Figure 2.23: Tower pro microservo MG995
2.3.6 Batteries

As a self operating car, the robot needed a power supply. One of the requirements is that the machine needs to be working for more than six hours, it is for that it needed either powerful batteries or a rechargeable battery but in a small size.

![Figure 2.24: 18650 3.7V Li-ion Battery](image)

On one hand, using these batteries instead of an external rechargeable battery secures that the device would obtain the necessary current. Because depending of the intake, the robot could need more or less batteries in parallel to supply the necessary current.

On the other hand, the rechargeable battery has more than one cycle of life.

2.3.7 Power supply Raspberry Pi

As mentioned above, the power supply of the robot is through batteries but the onboard computer needed a converter to transform the 6.7V from the batteries to the 5V needed. This converter is a DC-DC converter with an output in the form of a USB wire.

![Figure 2.25: Step-down DC-DC converter module](image)
2.3.8 Connections

To join the different modules, the robot needs wires. It uses interconnection cables between the Raspberry and the different modules, and to connect to the camera it uses a USB cable.

Assembling all the parts. The final prototype was the next one.

Figure 2.26: Car design high angle

Figure 2.27: Car design rear angle

Figure 2.28: Car design frontal angle
2.4 Line follower behavior

The first idea was to make a robot with two wheels, but we realised that with four wheels the device could maneuver more precisely.

The rear wheels are controlled by the DC motors but the front wheels are controlled by the servo motor. The robot drive following the next performance.

The rear wheels are the responsible of moving forward or backwards, the last option is in the case that the device does not detect the path. If the robot detects the line, the wheels make it move forward in a determined speed. This speed changes when the robot detects a deviation so it does not get off the route.

In the meantime, the front wheels are the responsible for changing the direction. The servo motor rotates the axis when the robot detects a deviation on the route.

The next image is a simple scheme of the behaviour of the robot and the process by which it decides in a specific scenario.

Figure 2.29: Line follower robot flowchart
The deviation is measured using the $centroid_x$ value and the $centroid_y$ value. Calculating those values, the robot knows if its position is on the right path or if it needs to change its trajectory. To change the trajectory, the machine calculates the angle of rotation needed.

![Diagram](image)

Figure 2.30: How calculate the angle

Depending on where the $centroid_x$ is, the difference between the ideal value and the real value change, in case of that the $centroid_x$ would be in the left side the subtract would be an absolute value, in the other case it would be a normal subtract.

Using the difference between the ideal value and the real value, we can calculate the angle of rotation.

\[
\Delta_x = \begin{cases} 
  c_x - c'_x & \text{if } centroid_x \geq ThresholdRight(x) \\
  |c_x - c'_x| & \text{if } centroid_x \leq ThresholdLeft(x) 
\end{cases} 
\] (2.6)

\[
hypotenuse = \sqrt{c_y^2 + \Delta_x^2} 
\] (2.7)

\[
\cosAngle = \frac{c_x}{hypotenuse} 
\] (2.8)
Once the cosine of the angle is calculated, the device can calculate the angle of rotation. In the case that the centroid$_x$ was on the left side, the angle would be the arc cosine of the cosine of the angle to get the angle. Whereas, if it was on the right side, the angle would be $180 - \text{arc cosine}$ of the cosine of the angle.

$$\text{angle} = \begin{cases} 
180 - \arccos(\cos\text{Angle}) & \text{if centroid}_x \geq \text{ThresholdRight}(x) \\
\arccos(\cos\text{Angle}) & \text{if centroid}_x \leq \text{ThresholdLeft}(x)
\end{cases}$$

(2.9)

### 2.5 PID Tracking Control Algorithm

The PID$^8$ was introduced in this project to make the robot follow the track more efficiently. The error between the centroid$_x$ and the desired value is processed by the PID controller[13], depending on the difference, it adjusts the speed of the rear wheels to correct the trajectory. The next image show how the control works:

The $x_d$ and $y_d$ are the desired values. Those values are subtracted with the values obtained with the onboard computer. The Raspberry Pi obtains the values using the camera, with that image from the camera the Raspberry calculates the angle of rotation so later it can calculate the approximated position of the LFR. The result of subtracting the ideal values to the real values shows the error on the position of the LFR.

Once the error is calculated the PID control changes the parameters to minimize the error on the next loop.

After the movement, the camera gets another image and the process starts again. Theoretically, the error ends in zero after $N$ loops, but really the error never arrives to be an exact zero.

---

$^8$Proportional Integral Derivative control
Chapter 3

Data acquisition

Another aim for this project was to capture the Wi-Fi signals and study the Wi-Fi coverage. To perform the task, we had to use an antenna, it was configured[14] into the board. The antenna has two functions, first it provides a communication between the user and the robot. Second, it receives the environmental Wi-Fi signals.

The point of this chapter is to explain the received signals function and how the antenna captures the signals to be processed and saved in a database by the computer.

At the beginning of the project, the idea was to use mobile phones as Wi-Fi antennae, but to make the task of data acquisition easier, we considered that using the Raspberry Pi antenna was the best choice.

This decision had advantages and disadvantages. The principal advantage was that it was easier to set up all the commands to acquire the signals. Otherwise, acquiring the signals with the same antenna, the one responsible of the communication between the user and the computer, produces noise in the Wi-Fi signals, all that was not important because we used a different frequency band for each task.
3.1 Signal detection

In this project, to detect signals\[15\] the program used command lines, those commands obtained information of the Wi-Fi signals. After some research and a few tries, the best option was the command \textit{iwlist} because it was the one that provided us more information on the signals around the antenna. Executing \textit{iwlist}, the command line showed the next information.

![iwlist command](image)

The image shows the parameters of one signal, but the command shows the Wi-Fi signals closer to the robot.

Once the program executed this command, it read every word to find the parameters of interest, for example the name of the signal, the quality, signal level, etc. With the found parameters, it created a database\[16\] using Microsoft Excel where the data was organized in different sheets, each signal had its own sheet.

The script ran during all the route once the program started, with no interference of the line follower robot.
Chapter 4

Communication user - robot

The last point was to establish a communication between the user and the robot with no cables. There are two types of communications, TCP or UDP, but that is discussed in the Appendix. After some attempts, the best option was to use the communication with the protocol TCP/IP\(^1\).

4.1 TCP/IP model

The TCP/IP model\(^{[17]}\) was used to net communication and describe a set of requirements to allow the device in the network. TCP/IP specifies how the data should be formatted, directed, transmitted, routed and received by the receiver.

To achieve a reliable data exchange between two devices, many separate procedures must be accomplished. The final result is a complex software, with a layer model or levels is easier to implement the software.

\[\text{Figure 4.1: Frame with the contours detected}\]

\(^1\)Transmission Control Protocol / International Protection
To program the connection between the user and the robot, we used Python Programming Language because it includes a class called `socket`, this class allows the users to create a communication with TCP protocol and UDP protocol\(^2\). In our project we needed a TCP protocol and we could achieve this tasks using Python’s sockets, called `SOCK_STREAM`. Theoretically, this socket guaranteed that the message sent would be received in the same order they were sent.

### 4.2 Graphic user interface

Once we had the robot working and the communication between the device and the user established, we had to develop the tool that would allow the robot to inform the user and the latter to control the device. To produce the tool we needed a software, and as we did before, we chose Python as it offers a library to make a GUI\([19, 20]\). As a way to improve the experience, the program used multithreading\([21]\), which means that every task runs parallel to others making the program work faster. In our GUI we have different options to modify the robot performance.

![Figure 4.2: Graphic user interface](image url)

- **Start button**: Send a message to start the line follower part. The robot starts to detect Wi-Fi information when the program runs, it is not necessary to start the line follower.
- **End button**: Send a message to end the program.
- **Run button**: Send a message to continue with the route. This button only can be used if the robot is stopped, in any other case, the button is disabled.
- **Stop button**: Send a message to stop the route.

\(^2\)User Datagram Protocol
• Speed Scrollbar: Send a value between 0 to 100, this value means the speed percentage within limits.

• Colour mode: Send a message with the name of the colour that it has to detect. There is the option None, this option disables the detector.

• Wi-Fi information window: Receives the Wi-Fi information of the environment and it is displayed in the screen.
Chapter 5

Budget

The project was about a self-operating mobile machine, the robot was a prototype and in this chapter we detail on the economical component of every part of the project. Doing this we could make an estimation of the number of hours dedicated on the thesis and the cost of the whole project. To build the project we bought the next components:

- Raspberry Pi 3
- Step-down DC-DC
- USB cable
- TF card
- USB camera
- Li-ion battery
- DC motor driver
- Battery holder
- Robot chassis

To calculate the budget of the project, we need to find the prices of every component and compare the price of manufacturing one unit with the price of a big number of units. The next table shows the cost\(^1\) of building the prototype.

\(^1\)All this prices were contrasted in Farnell[22], Mouser[23], TME[24], Robotshop[25], Adafruit[26] and DigiKey[27] web pages.
### Chapter 5. Budget

<table>
<thead>
<tr>
<th>Component</th>
<th>Number</th>
<th>Price 1u.(€)</th>
<th>Price 100u.(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 3</td>
<td>1</td>
<td>38.34</td>
<td>36.65</td>
</tr>
<tr>
<td>TF card and adapter</td>
<td>1</td>
<td>14.35</td>
<td>11.34</td>
</tr>
<tr>
<td>L298N</td>
<td>1</td>
<td>4.43</td>
<td>2.91</td>
</tr>
<tr>
<td>Step-down DC-DC</td>
<td>1</td>
<td>12.71</td>
<td>10.17</td>
</tr>
<tr>
<td>USB camera</td>
<td>1</td>
<td>8.59</td>
<td>7.70</td>
</tr>
<tr>
<td>Battery holder</td>
<td>1</td>
<td>1.23</td>
<td>0.573</td>
</tr>
<tr>
<td>USB cable</td>
<td>1</td>
<td>3.57</td>
<td>3.12</td>
</tr>
<tr>
<td>Li-ion battery</td>
<td>2</td>
<td>9.79</td>
<td>8.83</td>
</tr>
<tr>
<td>Li-ion battery charger</td>
<td>1</td>
<td>7.36</td>
<td>4.49</td>
</tr>
<tr>
<td>Car structure</td>
<td>1</td>
<td>33.32</td>
<td>32.81</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>133.69</strong></td>
<td><strong>118.59</strong></td>
</tr>
</tbody>
</table>

Table 5.1: Cost of the components.

The cost of one unit was 133.69€, but to manufacture 100 units the cost would be 118.59€. Usually, to compare the costs of manufacturing, we would compare the cost of 1 unit to building 1000 units, but in this case, in this kind of product we would not have such a big demand.

Regarding the software costs, Python is a free developer software.

In this case the team has only one member, so we evaluated the cost of a junior engineer, which meant more time than a group of engineers due to setbacks and the lack of knowledge. According to the studies, the average salary of a junior engineer in Spain is around 25000€/year\[28, 29\] (gross salary), approximately 1600€/month (net salary). That means 10€/hour.
The next diagram shows the duration of the project.

![Gantt diagram](image.png)

Figure 5.1: Gantt diagram

Taking into account an average of 4 hour per day of work and how much a junior engineer earns per hour, we can estimate the cost of the team. The project was 110 workdays, that means 440 hours of work. The salary of the team in this project costs \(4400\)€. Adding the costs of making one prototype, the total cost of the project\(^2\) was \(4533.69\)€.

\(^2\)When we talk about the whole project it means everything except the rent of an office and other expenses as water, electricity, taxes, etc.
Chapter 6

Conclusions and future work

6.1 Conclusions

6.1.1 Line Follower Robot

This prototype ends the first part of the project. Which means there are more steps to do in the future.

Currently, the prototype works right doing all the tasks proposed at the beginning of the project. The camera takes the image, the code analyzes it and calculates the centroids and decides the next movement to follow the line, it works with a good precision following the line without getting off. On the other side, the communication laptop-robot makes the decisions of the robot less fast, but the speed is not an important fact in this project. There is an issue with the code, the background subtracter function forces the user to start the program with the robot looking at a white surface.

6.1.2 Data acquisition

The antenna of the prototype is able to detect Wi-Fi signals in a range of frequency (3G or 5G). The test have been done with 3G but the goal is to test it with 5G.

The Wi-Fi antenna of the Raspberry Pi 3B is used to detect the signals, but at the same time it is used for the communication between the robot and the user, which could cause some interferences when the antenna has to detect the Wi-Fi signals at the same range that the robot uses to communicate, but when we want to work with a frequency we can use the other range to communicate. The code detects the signals of the environment and those signals are saved in a Excel document as database, where every signal has its own sheet.

6.1.3 Graphic User Interface

To communicate the user and the robot we used a Python library called Tkinter, which is easy to understand and includes a lot of functions. The graphic user interface connects the user with the onboard computer and establishes the communication using
sockets to make the connection. To do several tasks at the same time it uses multithreading. In that way, we achieve a faster and more secure connection.

The GUI allows the user to interact with the robot through the commands seen it. In the GUI we can change some parameters such as the speed or the state of the robot.

### 6.2 Future work

#### 6.2.1 Line Follower Robot

First, everything done in the prototype has to be added in the final design. Once everything is applied, due to the differences of size and weight, the designer will have to adjust some parameters to ensure the good performance of the robot, as the PID control.

Second, this project has more than one goal. This means that the robot will have more than one task. The designer will need a servo control PCA9685\[30\] to command more modules. In this part of the project, the robot only needs a few modules that you can handle with just the Raspberry Pi 3B. It is for that, that the servo control PCA9685 was not included in the design.

Third, to add another improvement the user could add an illumination system to avoid issues with the lighting in the room. Using this, we can quit the background subtracter function in the code, which causes some issues.

#### 6.2.2 Data acquisition

Another future task would be to process all the information caught during the experiment and process it to develop a coverage study. This task was not finished because of the short period of time to perform this part of the project and there was not enough time, also the team decided that it was not relevant compared to other chores.

It would be a good solution add two mobile phones as antennae to catch the signals. Every mobile phone would catch signals in a different range and in this way use the Wi-Fi antenna just to communicate the robot with the laptop of the user.

It would be a good solution to add two mobile phones as antennae to detect the signals. Every mobile phone would be exposed to signals in a different range and that way use the antenna of the computer only for the communication between the robot and the user.

Another option would be to use a laptop on the robot. Doing this, there is no necessity to communicate the robot with the user’s laptop. This solution means that the robot needs a large surface to hold the laptop.

As a possibility, it could add an indoor location[31, 32]. In that way, the robot after the experiment would be able to process the information and make a hot map with the data obtained.
6.2.3 Graphic User Interface

The next step depends on what it is described in the previous section. The principal idea is to change the configuration of the robot to hold a laptop and remove the distance communication with the user.

In case that it continues with the actual configuration, the GUI needs to optimize the communication to make it faster and more efficient. Moreover, the idea is to add more functions, that represent a change in the GUI adding more elements.
Bibliography

   http://playwithrobots.com/simple-line-follower-robot/

   http://opencv.org/

   http://http://docs.opencv.org/3.1.0/

   https://es.slideshare.net/AlanAguilarPerez/centroides-einercia

   http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/

   https://en.wikipedia.org/wiki/Lab_color_space

   https://en.wikipedia.org/wiki/Lab_color_space

   https://www.raspeberry.org/

   https://docs.python.org/2/

    RPI-CM-DATASHEET-V1_0.pdf

    Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900-1024x341.png

    https://learn.adafruit.com/
    playing-sounds-and-using-buttons-with-raspberry-pi/
    install-python-module-rpi-dot-gpio
http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html


https://www.calazan.com/
how-to-continuously-monitor-your-wi-fis-signal-strength-in-ubuntu/

https://openpyxl.readthedocs.io/en/default/


http://developeando.net/sockets-python/

http://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html

https://www.tutorialspoint.com/python/python_gui_programming.htm


[22] Farnell element14 (On line).
http://farnell.com/

http://www.mouser.es/

http://www.tme.eu/es/

http://www.robotshop.com/

https://www.adafruit.com/
[27] DigiKey (On line).
https://www.digikey.es/

http://noticias.universia.es/educacion/noticia/2015/06/30/1127468/
estudiar-ingenieria-telecomunicacion.html

https://www.indeed.es/salaries/Ingeniero/a-en-telecomunicaciones-Salaries

https://cdn-learn.adafruit.com/downloads/pdf/
adafruit-16-channel-servo-driver-with-raspberry-pi.pdf

http://www.cs.cmu.edu/~mmv/papers/10icra-joydeep.pdf


Appendix A

LAB Color Space

Color space defined by the CIE, based on one channel for Luminance (lightness) (L) and two color channels (a and b). One problem with the XYZ color system, is that colorimetric distances between the individual colors do not correspond to perceived color differences. For example, in the figure above, a difference between green and greenish-yellow is relatively large, whereas the distance distinguishing blue and red is quite small. The CIE solved this problem in 1976 with the development of the three-dimensional Lab color space (or CIELAB color space).

In this model, the color differences which you perceive correspond to distances when measured colorimetrically. The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b). The brightness (L) increases from the bottom to the top of the three dimensional model.

![Figure A.1: CIELAB color space](image)

This color space is better suited to many digital image manipulations than the RGB space, which is typically used in image editing programs.
Appendix B

Ramer Douglas Peucker algorithm

The purpose of the algorithm is, given a curve composed of line segments (which is also called a Polyline in some contexts), to find a similar curve with fewer points. The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve (i.e., the Hausdorff distance between the curves). The simplified curve consists of a subset of the points that defined the original curve.

![Figure B.1: Ramer–Douglas–Peucker algorithm](image)

The starting curve is an ordered set of points or lines and the distance dimension bigger than zero.

The algorithm recursively divides the line. Initially it is given all the points between the
first and last point. It automatically marks the first and last point to be kept. It then finds the point that is furthest from the line segment with the first and last points as end points; this point is obviously furthest on the curve from the approximating line segment between the end points. If the point is closer than the variable to the line segment, then any points not currently marked to be kept can be discarded without the simplified curve being worse than the variable.

If the point furthest from the line segment is greater than the variable from the approximation then that point must be kept. The algorithm recursively calls itself with the first point and the furthest point and then with the furthest point and the last point, which includes the furthest point being marked as kept.

When the recursion is completed a new output curve can be generated consisting of all and only those points that have been marked as kept.
Appendix C

Proportional Integral Derivative control

The PID control[33] is about a control loop feedback mechanism widely used in a variety of applications requiring continuously modulated control. The point of this method is calculate an error value as the difference between a desired point and a measured point and applies a correction based on proportional, integral and derivative terms. The next picture show the scheme of the control:

![Figure C.1: Scheme of a PID Control](image)

In our case, the controller calculates the current position, and then calculates the error based on the current position. It will then command the motors to take a hard turn, if the error is high or small turn, if the error is low. Basically, the magnitude of the turn taken will be proportional to the error. This is a result of the proportional control.

Even after this, if the error does not decrease approximately to zero, the controller will then increase the magnitude of the turn further and further over time until the centroid over the line. This is the result of the integral control.

In the process of centering over the line, the robot may overshoot the target position and move to the other side of the line where the above process is followed again. Thus the robot may keep oscillating about the line in order to center over the line. To reduce the oscillating effect over time, the derivate control is used. The proportional term is only a gain amplifier, and the derivate term is applied in order to improve the response
to disturbance, and also to compensate for phase lag at the controlled object. Example of pseudo code for the PID controller would be:

```
kp = 8
ki = 0.1
kd = 0.2
offset = 45
integral = 0
lastError = 0
derivative = 0
Loop forever:
    CameraValue = centroid_x
    error = 0 - CameraValue
    integral = integral + error
    derivative = error - lastError
    Turn = kp * error + ki * integral + kd * derivative
    powerA = Tp + Turn
    powerB = Tp - Turn
    MOTOR A direction = forward power = PowerA
    MOTOR B direction = forward power = PowerB
    lastError = error
End loop forever
```

PID controller requires the $k_p$, $k_i$ and $k_d$ factors to be set to match wheeled line follower robot’s characteristics and these values depends on robot structures, actuators, camera and other electronic circuits. There is no easy way to calculate the constants. It requires manual trial and error method until you get the desired behavior. We defined these factors using the trial and error theory. The steps to follow are the next:

- Start with low speed and setting constant values to 0.
- Then, try setting $k_p$ to a value of 1 and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly.
- Once the robot is able to follow the line, set $k_d$ value to 1 and then try increasing this value until you see less wobble.
- Once the robot is fairly stable at following the line, assign a value from 0.5 to 1 to $k_i$. If this value is too high, the robot will jerk left and right quickly. If it is too low, you won’t see any perceivable difference. Since integral is cumulative, the $k_i$ value has a significant impact.
- Once the robot is following the line with good accuracy, you can increase the speed and see if it is still able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.
Appendix D

UDP Protocol vs TCP Protocol

UDP is a protocol no oriented to connection. That means, when a machine A send a package to a machine B, the flow is unidirectional. The data transfer is realised without previously establish a connection with the machine B, the receiver (machine B) will receive the data without send a confirmation to the transceiver (machine A). That is because the data encapsulation sent through the protocol doesn’t allow send the information related to the sender. Therefore, the receiver will not know who is the transceiver, just his IP.

TCP is a protocol, unlike UDP, oriented to connection. When a machine A sends data to a machine B, machine B is informed of the received data and confirm the good reception. The CRC control of data intervene in the communication, it is based in a mathematical equation that allows to verify the information received. In that way, if the data are wrong, the TCP protocol approve that the receiver requests to the sender to resend the data.

The next table shows the different between the protocols
<table>
<thead>
<tr>
<th></th>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Not oriented to connection.</td>
<td>Oriented to connection.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Send data, but it is not based in connections, one program can send a lot of data packages and it is all.</td>
<td>Send data between two computers through Internet.</td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
<td>It is useful to applications that need a high reliability and the transmission time is not important.</td>
<td>It is useful to applications that need a fast data transmission. The fact of the no connection transfer make it useful to servers with a huge quantity of small requests of a high number of users.</td>
</tr>
<tr>
<td><strong>Order data package</strong></td>
<td>The data package is independent from the others, so there is no order.</td>
<td>Reorder the data in a determined form.</td>
</tr>
<tr>
<td><strong>Data speed</strong></td>
<td>It is faster due the protocol doesn’t verify the errors in the data.</td>
<td>It is slower than UDP.</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>There is no guarantee that the data package arrive to the destiny.</td>
<td>It make sure the reception of the sent data with out change the order of the packages.</td>
</tr>
<tr>
<td><strong>Weigh</strong></td>
<td>8 bits</td>
<td>20 bits</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>It is light due there is no order, verification of connection, etc.</td>
<td>It needs 3 packages to establish the connection before the transmission.</td>
</tr>
<tr>
<td><strong>Error checking</strong></td>
<td>It has a error verification, but there is no option of recover or correct thee data.</td>
<td>It has a error verification.</td>
</tr>
<tr>
<td><strong>Checksum</strong></td>
<td>Just to detect errors</td>
<td>Full</td>
</tr>
<tr>
<td><strong>Handshake</strong></td>
<td>It doesn’t have the recognition.</td>
<td>SYN, SYN-ACK, ACK.</td>
</tr>
</tbody>
</table>

Table D.1: UDP vs TCP.
Appendix E

Line Follower Robot

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# lineFollowerRobot.py
# David Torres Alcantud — 21 August 2017 — v.4.0
# Capture a picture of the environment and control the movements of the robot.

import cv2
import numpy as np
import sys
import time
import cv2
import numpy
import math
import RPi.GPIO as GPIO
from picamera.array import PiRGBArray
from picamera import PiCamera
from detectColor import DetectColor

def setup():
    global PWMaxis
    global PWMright
    global PWMrightDuty
    global PWMleft
    global PWMleftDuty
    global clock
    global enableColor
    global speed
    global kp
    global ki
    global kd
    global integral
    global lastError
    global derivative
    global capture
    global fgbg
    kp = 0
    ki = 0
    kd = 0
    lastError = 0
derivative = 0
integral = 0
speed = 30
clock = 0
enableColor = False

#The GPIO.BCM are the numbers after "GPIO" in the green rectangles around the outside of the below diagrams.
GPIO.setmode(GPIO.BCM)  # Pins by the Broadcom SOC channel number.
GPIO.setwarnings(False)

# Set up configuration of the wheels.
Servo GPIO pin.
GPIO.setup(9, GPIO.OUT)

motor GPIO pins.
GPIO.setup(16, GPIO.OUT)
GPIO.setup(20, GPIO.OUT)
PWM GPIO pins.
GPIO.setup(24, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
PWM starts at 0% dutycycle except the axis.
PWMright.start(0)
PWMrightDuty.start(0)
PWMleft.start(0)
PWMleftDuty.start(0)
PWMaxis.start(7)

# PWM starts at 0/18 * 90 + 2 = 7,
Enable motor pins.
GPIO.output(16, GPIO.HIGH)
GPIO.output(20, GPIO.HIGH)

# Set up the camera.
capture = cv2.VideoCapture(0)  # Read the video.
capture.set(3, 320.0)
capture.set(4, 240.0)
capture.set(5, 15)
for i in range (0,2):
    flag, trash = capture.read()  # Starting unwanted null value.

fgbg = cv2.BackgroundSubtractorMOG()
def end():
    desiredPosition = 90
    DC=1./18.*(desiredPosition)+2
    PWMaxis.ChangeDutyCycle(DC)
    time.sleep(0.5)
    PWMrigh.stop()
    PWMrighDuty.stop()
    PWMleft.stop()
    PWMleftDuty.stop()
    PWMaxis.stop()
    GPIO.cleanup()

#Calculate the angle of rotation of the axis.
def calcAngle(cx, cy):
    hypotenuse = math.sqrt(math.pow(cy, 2) + math.pow(cx, 2))
    cosAngle = cx/hypotenuse
    angle = math.acos(cosAngle)
    angle = math.degrees(angle)
    return angle

#Change the speed of the robot.
def setSpeed(percentage, enable):
    if enable == True:
        global speed
        oldSpeed = speed
        speed = oldSpeed * (int(percentage) / 100)
        PWMrighDuty.ChangeDutyCycle(speed)
        PWMleftDuty.ChangeDutyCycle(speed)

#Set the parameters to improve the behaviour of the robot.
def PIDcontrol(centroidX):
    global integral
    global derivate
    global lastError
    global kp
    global ki
    global kd
    error = 0
    turn = 0
    error = 160 - centroidX
    integral = integral + error
    derivative = error - lastError
    turn = (kp * error) + (ki * integral) + (kd * derivative)
    kp = kp + 10
    lastError = error
    return turn

def initialize(enable, color):
    global clock
    global speed
    global enableColor
    global capture
    global fgbg
    programStart = False

    #Set the value of how much time the user wants the duration of the experiment.
if enable == True and clock < 60:
    start = time.time()
    # Processing of the image.
    # ----------------------------------
    for i in range (0,2):
        flag, trash = capture.read() # Starting unwanted null value
        flag, frame = capture.read()
        rows = frame.shape[0]
        cols = frame.shape[1]
        roi = frame[10:2∗rows/3, rows/12:cols−20]  # Region of interest.
        # Detect if there is some color and if it detect
        # the color then sleep the car.
        # ----------------------------------
        if color != ":
            enableColor = DetectColor().detectShapeColor(roi, color)
            if enableColor == True:
                time.sleep(10)
    # Detect the black line.
    # ----------------------------------
    grayscale = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)  # Convert the frame to grayscale.
    blur = cv2.GaussianBlur(grayscale, (5,5), 0)  # Blur the grayscale, Gaussian kernel size is (5,5) and desviation of the kernel is 0.
    fgmask = fgbg.apply(blur, learningRate=0.001)
    retValInvert, thresholdInvert1 = cv2.threshold(fgmask, 0, 255, cv2.THRESH_BINARY_INV)  # Invert the pixels of the image frame. The threshold value which is used to classify is 127.
    retValNoise, thresholdNoise = cv2.threshold(thresholdInvert1, 20, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  # Remove the noise using threshold. The threshold value which is used to classify is 32.
    retValInvert, thresholdInvert2 = cv2.threshold(thresholdNoise, 0, 255, cv2.THRESH_BINARY_INV)  # Invert the pixels of the image frame. The threshold value which is used to classify is 127.
    erodeImg = cv2.erode(thresholdInvert2, np.ones((5,5), np.uint8), iterations = 1)
    dilateImg = cv2.dilate(erodeImg, np.ones((5,5), np.uint8), iterations = 1)
    contours, hierarchy = cv2.findContours(dilateImg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  # Find the contours.
    # ----------------------------------
    # Find the area, moment and centroids of contour.
    # ----------------------------------
    for contour in contours:
        programStart = True
        if contour is not None:
            area = cv2.contourArea(contour)  # Return the area and the number of non-zero pixels.
            if area >= 100:
                moment = cv2.moments(contour)
                centroidX = int(moment['m10']/moment['m00'])
                centroidY = int(moment['m01']/moment['m00'])
if centroidX < 134:
    difCentrX = abs(centroidX - 160)
    desiredPosition = calcAngle(difCentrX, centroidY)
    DC=1./18.*(desiredPosition) + 2
    PWMleftDuty.ChangeDutyCycle(speed)
    PWMrightDuty.ChangeDutyCycle(speed)
    PWMleft.ChangeDutyCycle(0)
    PWMright.ChangeDutyCycle(0)
    PWMaxis.ChangeDutyCycle(DC)
    time.sleep(0.05)
elif centroidX > 186:
    difCentrX = centroidX - 160
    desiredPosition = calcAngle(difCentrX, centroidY)
    DC=1./18.*(180 - desiredPosition) + 2
    PWMleftDuty.ChangeDutyCycle(speed)
    PWMrightDuty.ChangeDutyCycle(speed)
    PWMright.ChangeDutyCycle(0)
    PWMleft.ChangeDutyCycle(0)
    PWMaxis.ChangeDutyCycle(DC)
    time.sleep(0.05)
else:
    desiredPosition = 90
    DC=1./18.*(desiredPosition) + 2
    PWMrightDuty.ChangeDutyCycle(speed)
    PWMleftDuty.ChangeDutyCycle(speed)
    PWMright.ChangeDutyCycle(0)
    PWMleft.ChangeDutyCycle(0)
    PWMaxis.ChangeDutyCycle(DC)
    time.sleep(0.05)

if not contours and programStart == True:
    desiredPosition = 90
    DC=1./18.*(desiredPosition)+2
    PWMrightDuty.ChangeDutyCycle(0)
    PWMleftDuty.ChangeDutyCycle(0)
    PWMright.ChangeDutyCycle(speed)
    PWMleft.ChangeDutyCycle(speed)
    PWMaxis.ChangeDutyCycle(DC)
    time.sleep(0.05)

end = time.time()
clock = clock + (end-start)

elif enable == False:
    #If the enable is zero put the duty 0 to stop the wheels
    PWMrightDuty.ChangeDutyCycle(0)
    PWMleftDuty.ChangeDutyCycle(0)
    PWMright.ChangeDutyCycle(0)
    PWMleft.ChangeDutyCycle(0)

if __name__ == "__main__":
    setup()
    initialize(True, "")
    end()
Appendix F

Detection of an object

#!/usr/bin/env python
# −∗− coding: utf-8 −∗−
#
# detectColor.py
# David Torres Alcantud – 21 August 2017 – v.2.0
# Detect if there is an object with the shape and the color choose.

from shapeDetector import ShapeDetector
from colorLabeler import ColorLabeler
import imutils
import cv2

class DetectColor:
    def __init__(self):
        pass

    def detectShapeColor(self, image, col):
        enable = False
        #resize it to a smaller factor so that the shapes can be approximated better.
        resized = imutils.resize(image, width = 300)
        ratio = image.shape[0] / float(resized.shape[0])

        #blur the resized image slightly, then convert it to both grayscale and the L*a*b color spaces.
        blur = cv2.GaussianBlur(resized, (5, 5), 0)
        gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
        lab = cv2.cvtColor(blur, cv2.COLOR_BGR2LAB)
        thresh = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)[1]

        #find contours in the thresholded image.
        contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        if imutils.is_cv2():
            contours = contours[0]
        else:
            contours = contours[1]

        #initialize the shape detector and color labeler.
        sd = ShapeDetector()
        cl = ColorLabeler()
APPENDIX F. DETECTION OF AN OBJECT

#loop over the contours.
for contour in contours:
    #compute the center of the contour.
    M = cv2.moments(contour)
    try:
        cX = int((M["m10"] / M["m00"])) * ratio
        cY = int((M["m01"] / M["m00"])) * ratio
    except ZeroDivisionError:
        pass

    #detect the shape of the contour and label the color.
    shape = sd.detect(contour)
    color = cl.label(lab, contour)

    #multiply the contour (x, y)−coordinates by the resize ratio, draw the contours
    → and the name of the shape and labeled color on the image.
    contour = contour.astype("float")
    contour *= ratio
    contour = contour.astype("int")
    text = "{} {}".format(color, shape)
    #Detect if it is the stop point
    if color == col and shape == "rectangle":
        enable = True
    return enable
Appendix G

Detection of shapes on the track

#!/usr/bin/env python
# coding: utf-8
#
# shapeDetector.py
# David Torres Alcantud — 21 August 2017 — v.1.0
# Detect the shape of the objects in a picture.

import cv2

class ShapeDetector:
    def __init__(self):
        pass

    def detect(self, cnt):
        # initialize the shape name and approximate the contour.
        shape = "unidentified"
        perimeter = cv2.arcLength(cnt, True)
        approx = cv2.approxPolyDP(cnt, 0.04 * perimeter, True)

        # if the shape is a triangle, it will have 3 vertices.
        if len(approx) == 3:
            shape = "triangle"

        # if the shape has 4 vertices, it can be square or rectangle
        elif len(approx) == 4:
            # compute the bounding box of the contour and use the bounding box to compute the aspect ratio.
            (x, y, w, h) = cv2.boundingRect(approx)
            aspRat = w / float(h)

            if aspRat >= 0.95 and aspRat <= 1.05:
                shape = "square"
            else:
                shape = "rectangle"

        # otherwise, it assume the shape is a circle
        else:
            shape = "circle"
return shape
Appendix H

Detection of color on the track

#!/usr/bin/env python
#−∗− coding: utf−8 −∗−
#
# colorLabeler.py
# David Torres Alcantud − 21 August 2017 − v.1.0
# Detect the color of the object and put him the name.

from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np
import cv2

class ColorLabeler:
    def __init__(self):
        #initialize the colors dictionary, containing the color name as the key and the RGB tuple as the value.
        colors = OrderedDict({
            "red": (255,0,0),
            "green": (0,255,0),
            "blue": (0,0,255),
            "black": (0, 0, 0),
            "white": (255, 255, 255)}
        )

        #allocate memory for the L*a*b image, then iniatialize the color names list.
        self.lab = np.zeros((len(colors), 1, 3), dtype = "uint8")
        self.colorNames = []

        #loop over the colors dictionary.
        for (i, (name, rgb)) in enumerate(colors.items()):

            #update the L*a*b array and the color names list.
            self.lab[i] = rgb
            self.colorNames.append(name)

        #convert the L*a*b array from the RGB color space to L*a*b.
        self.lab = cv2.cvtColor(self.lab, cv2.COLOR_RGB2LAB)

    def label(self, image, cnt):

        #construct a mask for the contour, then compute the average L*a*b value for the masked region.
        mask = np.zeros(image.shape[:2], dtype = "uint8")

        cv2.drawContours(mask, [cnt], −1, 255, −1)

        #erode the mask to ensure statistics are only being computed for the masked region and that no background is included.
APPENDIX H. DETECTION OF COLOR ON THE TRACK

```python
mask = cv2.erode(mask, None, iterations = 2)

# calculate teh averages for each L*a*b.
mean = cv2.mean(image, mask = mask)[3]

minDist = (np.inf, None)

# loop over the known L*a*b color values
for (i, row) in enumerate(self.lab):
    # compute the distance between the current L*a*b color value and the mean of
    # the image.
    d = dist.euclidean(row[0], mean)

    # if the distance is smaller than the current distance, then update the
    # bookkeeping variable.
    if d < minDist[0]:
        minDist = (d, i)

# return the name of the color with the smallest distance.
return self.colorNames[minDist[1]]
```
Appendix I

Data acquisition

#!/usr/bin/env python
# −∗− coding: utf-8 −∗−
#
# wifiDetector.py
# David Torres Alcantud − 21 August 2017 − v.1.0
# Parses the output of iwlist scan into a table and write it into an Excel document.

import sys
import subprocess
import datetime
import os
import numpy
import time
from openpyxl import Workbook
from openpyxl import load_workbook

interface = "wlan0"

global startTime
startTime = datetime.datetime.now().time()

def get_name(cell):
    return matching_line(cell,"ESSID:")[1:-1]

def get_quality(cell):
    quality = matching_line(cell,"Quality=").split()[0].split('/')
    return str(int(round(float(quality[0]) / float(quality[1]) * 100))) + " %"

def get_channel(cell):
    return matching_line(cell,"Channel:")

def get_signal_level(cell):
    return matching_line(cell,"Signal_level=")[1]

def get_encryption(cell):
    enc=""
    if matching_line(cell,"Encryption_key:" ) == "off":
        enc="Open"
    else:
        for line in cell:
            print(line)
matching = match(line,"IE:"
if matching!=None:
wpa=match(matching,"WPA_Version:")
if wpa!=None:
    enc="WPA_v."+wpa
    if enc="":
        enc="WEP"
return enc

def get_address(cell):
    return matching_line(cell,"Address:")

# Dictionary of rules that will be applied to the description of each cell. The key will be the name of the column in the table. The value is a function defined above.

rules = {
    "Name":get_name,
    "Quality":get_quality,
    "Channel":get_channel,
    "Encryption":get_encryption,
    "Address":get_address,
    "Signal":get_signal_level
}

# Choose the way of sorting the table. sort_by should be a key of the dictionary rules.
def sort_cells(cells):
    sortby = "Quality"
    reverse = True
    cells.sort(None, lambda e:el[sortby], reverse)

# Choose which columns to display here, and most importantly in what order.
columns = ["Name", "Address", "Quality", "Signal", "Channel", "Encryption"]

# Returns the first matching line in a list of lines.
def matching_line(lines, keyword):
    for line in lines:
        matching=match(line,keyword)
        if matching!=None:
            return matching
    return None

# If the first part of line matches keyword, returns the end of that line. Otherwise returns None
def match(line,keyword):
    line=line.lstrip()
    length=len(keyword)
    if line[:length] == keyword:
        return line[length:]
    else:
        return None

# Applies the rules to the bunch of text describing a cell and returns the corresponding dictionary
def parse_cell(cell):
    parsed_cell={}
    for key in rules:
        rule=rules[key]
        parsed_cell.update({key:rule(cell)})
return parsed_cell

# Detect the last row in the Excel document to continue writing.

def detectLastRow(ws):
    lastRowSheet = ws.max_row
    lastRow = 1
    for r in range(1, lastRowSheet):
        if not ws.cell(row = r, column = 1).value is None:
            lastRow = r + 1
        else:
            break
    lastRow = lastRow + 1
    return lastRow

def justifyTable(table):
    widths = map(max, map(lambda l:map(len,l), zip(*table)))  # functional magic
    justified_table = []
    for line in table:
        justified_line = []
        for i, el in enumerate(line):
            justified_line.append(el.ljust(widths[i]+2))
        justified_table.append(justified_line)
    return justified_table

def obtainTable(cells):
    table = [columns]
    for cell in cells:
        cell_properties = []
        for column in columns:
            cell_properties.append(cell[column])
        table.append(cell_properties)
    return table

def differenceTime(startTime, actualTime):
    form = "%H:%M:%S"
    h1 = datetime.datetime.strptime(startTime, form)
    h2 = datetime.datetime.strptime(actualTime, form)
    time = h2 - h1
    return time

def obtainExcel(table, wb):
    global startTime
    try:
        ws = wb.active
        table[0].append("Time")  # Add the time to the file.
        for infoTable in table:
            if len(infoTable) < 7:
                actualTime = datetime.datetime.now().time()
                time = differenceTime(str(startTime).split('.')[0], str(actualTime).split('.')[0])
                infoTable.append(str(time))
            found = False
            for sheet in wb.sheetnames:
                if infoTable[0] == sheet:
                    ws = wb.get_sheet_by_name(infoTable[0])
                    found = True
if found == False:
    ws = wb.create_sheet(infoTable[0])
    col = 1
    for info in table[0]:
        ws.cell(row = 1, column = col).value = info
        col += 1
    col = 1
    row = detectLastRow(ws)
    for info in infoTable:
        ws.cell(row = row, column = col).value = info
        col += 1
except IndexError:
    pass

# Delet the unwished sheets.
try:
    delet = wb.get_sheet_by_name(u'Sheet')
    wb.remove_sheet(delet)
except KeyError:
    pass
try:
    delet = wb.get_sheet_by_name(u'Name')
    wb.remove_sheet(delet)
except KeyError:
    pass

# Delet the parameter Time from the sheets.
try:
    del table[0][−1]
except KeyError:
    pass

return wb

def obtainSignal():
cells=[[]]
parsed_cells=[]
time = datetime.datetime.now()
day = time.day
month = time.month
year = time.year

pathFolder = "%/home/pi/Downloads/DataBase"
pathFile = "%/home/pi/Downloads/DataBase/DataBase" + str(year) + "−" + str(month) + "−" + str(day) + ".xlsx"

if os.path.exists(pathFolder) == False:
    os.makedirs(pathFolder)
if os.path.exists(pathFile) == False:
    wb = Workbook()
else:
    wb = load_workbook(pathFile)

proc = subprocess.Popen(["iwlist", interface, "scan"], stdout=subprocess.PIPE,
universal_newlines=True)
out, err = proc.communicate()

for line in out.split("\n"):
    cell_line = match(line,"Cell_")
    if cell_line != None:
        cells.append([[]])
        line = cell_line[-27:]
        cells[-1].append(line.rstrip())

for cell in cells:
    parsed_cells.append(parse_cell(cell))

sort_cells(parsed_cells)

# Variable with all the information for the GUI
table = obtainTable(parsed_cells)

# Justify the values for the GUI
justTab = justifyTable(table)

# Create the document with all the information.
wb = obtainExcel(table, wb)

# Save the Excel document.
wb.save(pathFile)

# To send the information we need a string, the we have to convert the list to a string.
info = ""
for signals in justTab:
    for signal in signals:
        info = info + signal
        info = info + "\n"

    time.sleep(3)

return info

if __name__ == "__main__":
    data = obtainSignal()
Appendix J

Communication User - Laptop

#!/usr/bin/env python
#
# tcpServer.py
# David Torres Alcantud — 21 August 2017 — v.2.0
# Establish the TCP/IP communication.

import RPi.GPIO as GPIO
import time
import Queue
from Queue import *
from socket import *
import lineFollowerRobot
import wifiDetector
from myThreadLineFollower import MyThreadLineFollower
from myThreadWifiDetector import MyThreadWifiDetector

ctrlCmd = ['run', 'exit', 'stop', 'start', 'speed', 'color']

busnum = 1  # Edit busnum to 0, if you use Raspberry Pi 1 or 0.

HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
tcpSerSock = socket(AF_INET, SOCK_STREAM)  # Create a socket.
tcpSerSock.bind(ADDR)  # Bind the IP address and port number of the server.
tcpSerSock.listen(5)  # Number the connections permitted at one time, when the connections are full, others will be rejected.

# Setup line follower robot class.
lineFollowerRobot.setup()

# Create a queue where save the values
q = Queue()

data = ''
while data != ctrlCmd[1]:
    print 'Waiting for connection...'

xxvii
# Waiting for connection. Once receiving a connection, the function accept() returns a separate
client socket for the subsequent communication.
By default, the function accept() is a blocking one which means it is suspended before the
connection comes.
tcpCliSock, addr = tcpSerSock.accept()
enable = False # Indicate the run/stop of the car.
color = "" # Variable to indicate if the robot will make stops or not.
print addr[0] #Print the IP addres of the client connected with the server.
while data != 'exit':
    try:
        tcpCliSock.settimeout(1)
data = tcpCliSock.recv(BUFSIZ) #Receive data sent from the client.
    except timeout:
        pass

# Analyze the command received and choose the option to move the car.
if not data:
    break
if data == ctrlCmd[0]:    #run.
enable = True
elif data == ctrlCmd[1]:  #exit.
enable = False
lineFollowerRobot.end()
elif data == ctrlCmd[2]:  #stop.
enable = False
elif data == ctrlCmd[3]:  #start.
enable = True
elif data[0:5] == ctrlCmd[4]:  #speed.
    #The scale is so slow then accumulate the values and don’t send the correct
    #value.
    spd1 = data[−6:]
    spd2 = data[−7:]
    spd3 = data[−8:]
    if spd1[0:5] == ctrlCmd[4]:
        tmp = data[−1:]
elif spd2[0:5] == ctrlCmd[4]:
        tmp = data[−2:]
elif spd3[0:5] == ctrlCmd[4]:
        tmp = data[−3:]
else:
    tmp = 0
    speed = int(tmp)
if speed <10:
    speed = 10
elif speed > 90:
    speed = 90
lineFollowerRobot.setSpeed(speed, enable)
elif data[0:5] == ctrlCmd[5]:  #color.
    numLen = len(data) - len(ctrlCmd[5])
    if numLen == 2:
        tmp = data[−1:]  #Take the number for the option.
        option = int(tmp)
        if option == 1:
            color = ""
elif option == 2:
            color = "green"
elif option == 3:
color = "red"

elif option == 4:
    color = "blue"

else:
    color = "None"

else:
    print 'Command Error! Cannot recognize command: ' + str(data)

# Create new threads.
threadLineFollower = MyThreadLineFollower(enable, color)
threadWifiDetector = MyThreadWifiDetector(q)

# Start new Threads.
threadLineFollower.start()
threadWifiDetector.start()

try:
    tcpCliSock.send(q.get())
except error:
    pass

tcpSerSock.close()
Appendix K

Line follower Robot multi threading

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# myThreadLineFollower.py
# David Torres Alcantud – 21 August 2017 – v.1.0
# Create a thread for the line follower robot script.

import threading
import time
import lineFollowerRobot

class MyThreadLineFollower(threading.Thread):
    def __init__(self, enable, color):
        threading.Thread.__init__(self)
        self.enable = enable
        self.color = color

    def run(self):
        lineFollowerRobot.initialize(self.enable, self.color)
```

xxx
Appendix L

Data acquisition Robot multithreading

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# myThreadWifiDetector.py
# David Torres Alcantud – 21 August 2017 – v.1.0
# Create a thread for the wifi detector script.

import threading
import time
import wifiDetector

class MyThreadWifiDetector(threading.Thread):
    def __init__(self, q):
        threading.Thread.__init__(self)
        self.q = q

    def run(self):
        signal = wifiDetector.obtainSignal()
        self.q.put(signal)
```

xxxi
Appendix M

Graphic user interface

#!/usr/bin/python
#
# coding: iso-8859-1
#
# GUI.py
# David Torres Alcantud – 21 August 2017 – v.3.0
# Graphic user interface that interacts with the user.

import decimal
import commands
from socket import *
from tkinter import *
from PIL import ImageTk, Image

root = Tk()  # Create a top window
root.title('Line Follower Robot')

HOST = '128.130.250.16'  # Server (Raspberry Pi) IP address.
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
tcpCliSock = socket(AF_INET, SOCK_STREAM)  # Create a socket.
tcpCliSock.connect(ADDR)

labelVariable = StringVar()  # Speed variable.
path = 'C:\Users\Datoal\Downloads\logo.PNG'  # Path where the logo is.
image = Image.open(path)
image = image.resize((150, 100), Image.ANTIALIAS)
img = ImageTk.PhotoImage(image)  # Variable that contains the logo.
var = DoubleVar()  # Variable with the velocity.
spd = 10
color = IntVar()

# ===============================================
# Create labels.
# ===============================================
labelTurn = LabelFrame(root, text = "Turn on/off", height = 10, width = 15)
labelControl = LabelFrame(root, text = "Robot Control", height = 5, width = 10)
labelScale = LabelFrame(labelControl, text = "Speed (%)", height = 5, width = 10)
labelHotSpot = LabelFrame(root, text = "Hotspot list", height = 5, width = 10)
labelColor = LabelFrame(root, text = "Color selection", height = 5, width = 10)
labelSpeed = Label(labelScale, textvariable = labelVariable)
labelImage = Label(root, image = img)

# Label layout.
labelTurn.grid(column = 0, row = 1, rowspan = 2, padx = 20)
labelControl.grid(column = 1, row = 1, rowspan = 2, columnspan = 2, padx = 20)
labelScale.grid(column = 2, row = 1, rowspan = 3, padx = 20)
labelHotSpot.grid(column = 1, row = 3, columnspan = 2)
labelColor.grid(column = 0, row = 3)
labelSpeed.grid(column = 2, row = 3, columnspan = 1)
labelImage.grid(column = 0, row = 0)

# Create buttons.
buttonRun = Button(labelControl, text = 'Run', state = 'disable', command =
    lambda:
    OnButtonClickRun(buttonRun), height = 2, width = 10)
buttonStop = Button(labelControl, text = 'Stop', state = 'disable', command =
    lambda:
    OnButtonClickStop(buttonRun), height = 2, width = 10)
buttonStart = Button(labelTurn, text = 'Start', command =
    lambda: OnButtonStart(OnButtonClickStart(buttonStart, buttonStop), height = 2, width = 10))
buttonExit = Button(labelTurn, text = 'Exit', command =
    lambda: OnButtonClickExit(), height = 2, width = 10)

# Button layout.
buttonRun.grid(column = 1, row = 1, pady = 20, padx = 20)
buttonStop.grid(column = 1, row = 2, pady = 20)
buttonStart.grid(column = 0, row = 1, pady = 20)
buttonExit.grid(column = 0, row = 2, pady = 20, padx = 20)

#Print the logo in the GUI.
labelImage.photo = img

# The functions of every button.
def OnButtonClickStart(button, buttonStop):
    #Modify the state of the buttons.
    button.config(state = 'disable')
    buttonStop.config(state = 'normal')
    #Update the state to change the state.
    button.update()
    buttonStop.update()
    tcpCliSock.send('start')

def OnButtonClickExit():
    tcpCliSock.send('exit')
    tcpCliSock.close()
    root.destroy()

def OnButtonClickStop(button):
    #Modify the state of the buttons.
APPENDIX M. GRAPHIC USER INTERFACE

```python
button.config(state = 'normal')
# Update the state to change the state.
button.update()

tcpCliSock.send('stop')

def OnButtonClickRun(button):
    # Modify the state of the buttons.
    button.config(state = 'disable')
    # Update the state to change the state.
    button.update()

tcpCliSock.send('run')

def changeSpeed(ev=None):
    tmp = 'speed'
    global spd
    spd = speed.get()
    data = tmp + str(spd)  # Change the integers into strings and combine them with the string 'speed'.
    print 'sendData= %s' % data
    tcpCliSock.send(data)  # Send the speed data to the server(Raspberry Pi)

def SpeedValue(ev=None):
    global spd
    spd = scale.get()  # Obtain the value of the scale
    data = 'speed' + str(spd)
    tcpCliSock.send(data)  # Send the speed data to the server(Raspberry Pi)

# ================
# Create scale.
# ================

scale = Scale(labelScale, variable = var, from_=100, to=0, command = SpeedValue)
scale.set(50)

# ================
# Scale layout.
# ================
scale.grid(column = 2, row = 1, rowspan = 2)

def HotSpotValues():
    listHotSpot.delete(0, END)
    data = tcpCliSock.recv(BUFSIZ)
pos = 1
    signals = data.split("\n")
    for signal in signals:
        listHotSpot.insert(pos, signal)
pos += 1
    root.after(1000, HotSpotValues)

# ================
# Create list.
# ================

listHotSpot = Listbox(labelHotSpot, width = 70)
root.after(1000, HotSpotValues)

xxxiv
# List layout.
# ==============================================================
listHotSpot.grid(column = 1, row = 3)

def checkColor():
    data = "color" + str(color.get())
    tcpCliSock.send(data)

# ==============================================================
# Create radio buttons.
# ==============================================================
RadioButtonNone = Radiobutton(labelColor, text = "None", variable = color, value = 1, command = lambda: checkColor)
RadioButtonGreen = Radiobutton(labelColor, text = "Green", variable = color, value = 2, command = lambda: checkColor)
RadioButtonRed = Radiobutton(labelColor, text = "Red", variable = color, value = 3, command = lambda: checkColor)
RadioButtonBlue = Radiobutton(labelColor, text = "Blue", variable = color, value = 4, command = lambda: checkColor)

# ==============================================================
# List layout.
# ==============================================================
RadioButtonNone.grid(column = 0, row = 3, sticky=W)
RadioButtonGreen.grid(column = 0, row = 4, sticky=W)
RadioButtonRed.grid(column = 0, row = 5, sticky=W)
RadioButtonBlue.grid(column = 0, row = 6, sticky=W)

def initialize():
    root.mainloop()

if __name__ == "__main__":
    initialize()